Mitglied der Helmholtz-Gemeinschaft

Lawrence Livermore National Laboratory

JÜLICH FORSCHUNGSZENTRUM

Massively Parallel Loading

Wolfgang Frings[*], Dong Ahn[**], Todd Gamblin[**], Matt Legendre[**], Bronis de Supinski[**], Felix Wolf[***]

June 14th 2013

27th International Conference on Supercomputing, Eugene, OR

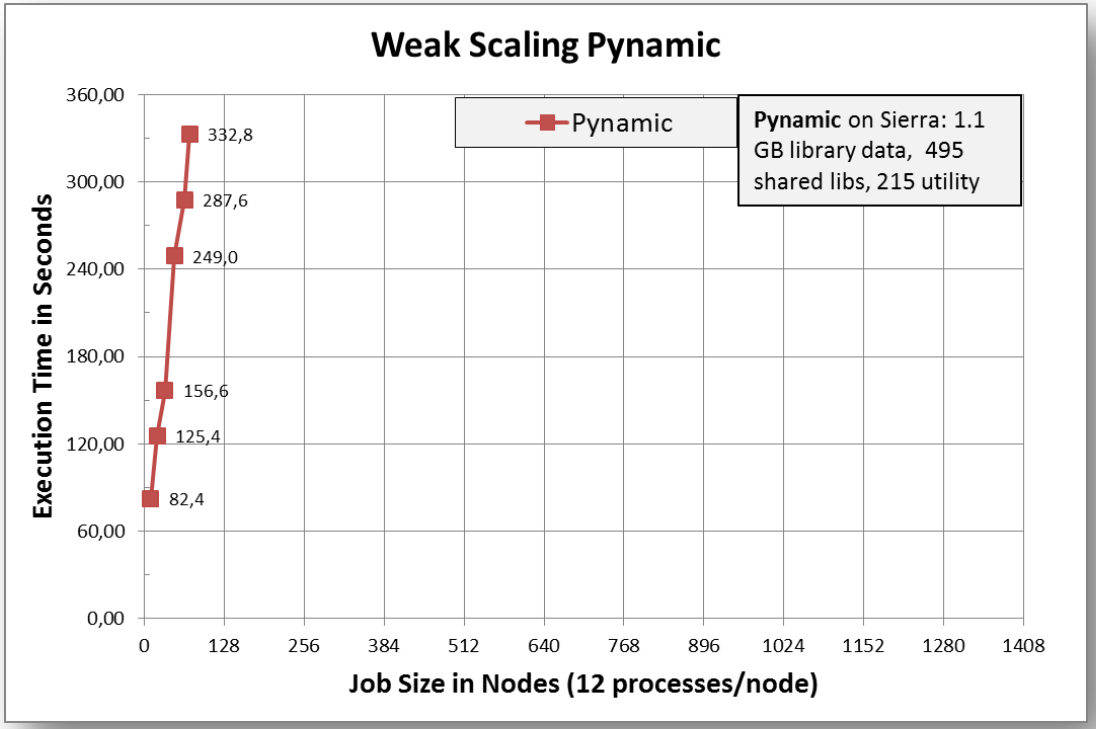
* Jülich Supercomputing Centre
** Lawrence Livermore National Laboratory
*** German Research School for Simulation Sciences

LLNL-PRES-638575
This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC

Dynamic Linking and Loading Causes Major Disruption at Large Scale

- **Multi-physics applications at LLNL**
  - 848 shared library files
  - Load time on BG/P:
    2k tasks → 1 hour
    16k tasks → 10 hours

- **Pynamic**
  - LLNL Benchmark
  - Loads shared libraries and python files
  - 495 shared objects → 1.1 GB



Pynamic running on LLNL Sierra Cluster
1944 nodes, 12 tasks/node,
NFS and Lustre file system

Challenges Mainly Arise from File Access Storms

- Caused by dynamic linker
  - **searching** and
  - **loading** dynamic linked libraries

- Formulas:

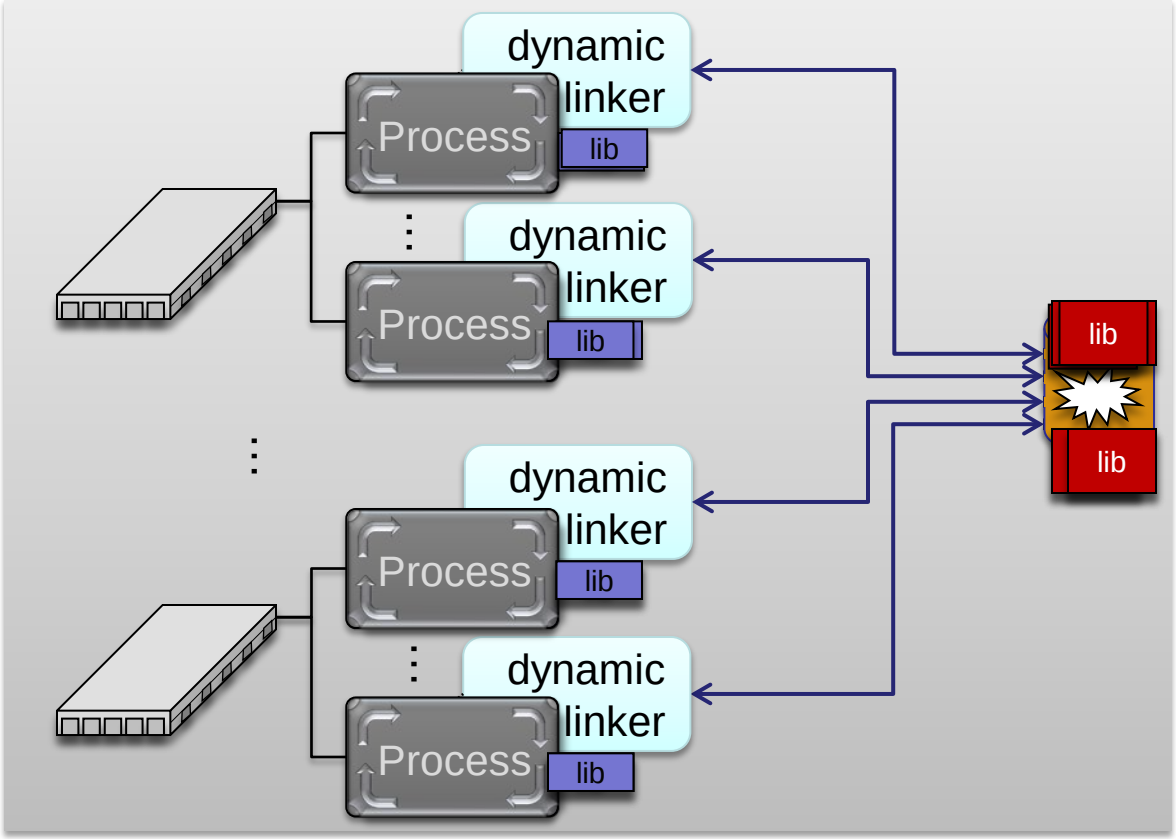
File metadata operations:
# of tests = # of processes x # of locations x # of libraries

File read operations:
# of reads = # of processes x # of libraries

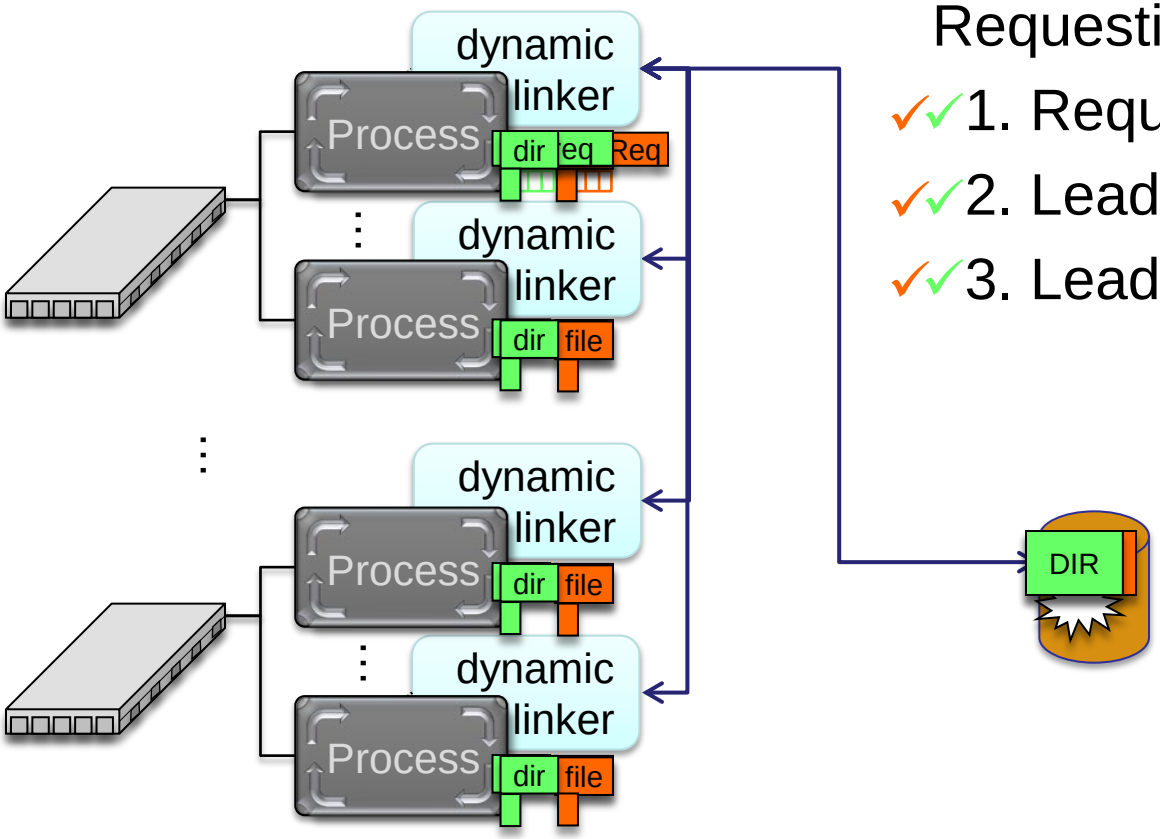
- Example: Pynamic Benchmark on Sierra-Cluster
  - serial (1 task): 5,671 open/stat calls
  - parallel (23,328 tasks) : 132,293,088 open/stat calls

File Access is uncoordinated!

- Loading is nearly unchanged since 1964 (MULTICS)
- ld-linux.so uses serial POSIX file operations that are not coordinated among process.



How SPINDLE Works



Requesting dir/file:

✓✓1. Request from leader

✓✓2. Leader reads from disk

✓✓3. Leader distributes to peers

File metadata operations:
# of tests = # of locations

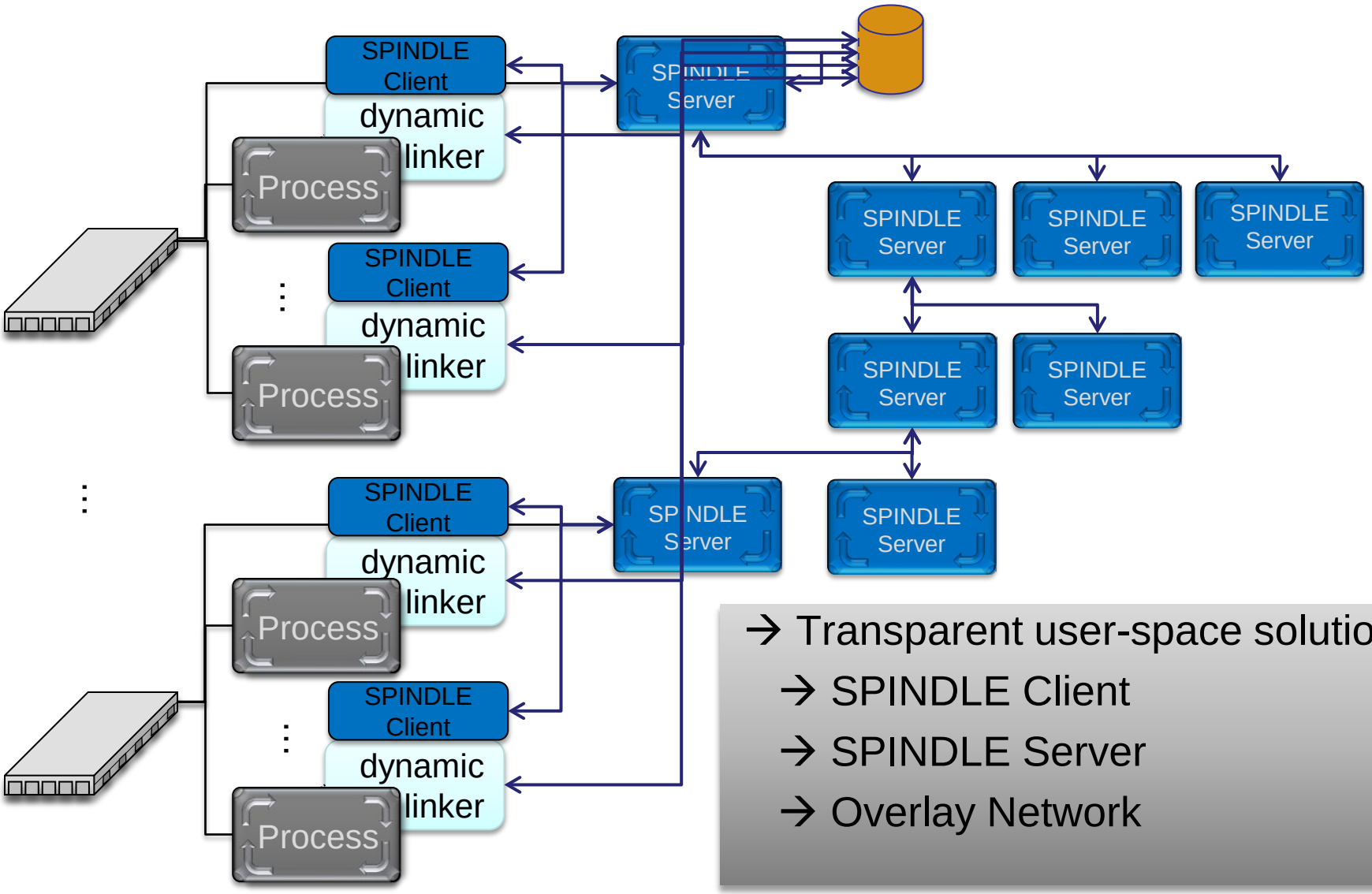
File read operations:
# of reads = # of libraries

SPINDLE: *Scalable Parallel Input Network for Dynamic Load Environments*

Overview

- Design & Implementation:
  - Components of SPINDLE
  - Interaction with the dynamic linker (client-side)
  - Strategies for caching and data distribution (Pull- or Push-Model)
  - Communication between SPINDLE server (overlay network)
- Performance
- Memory Usage
- Usage and features of SPINDLE

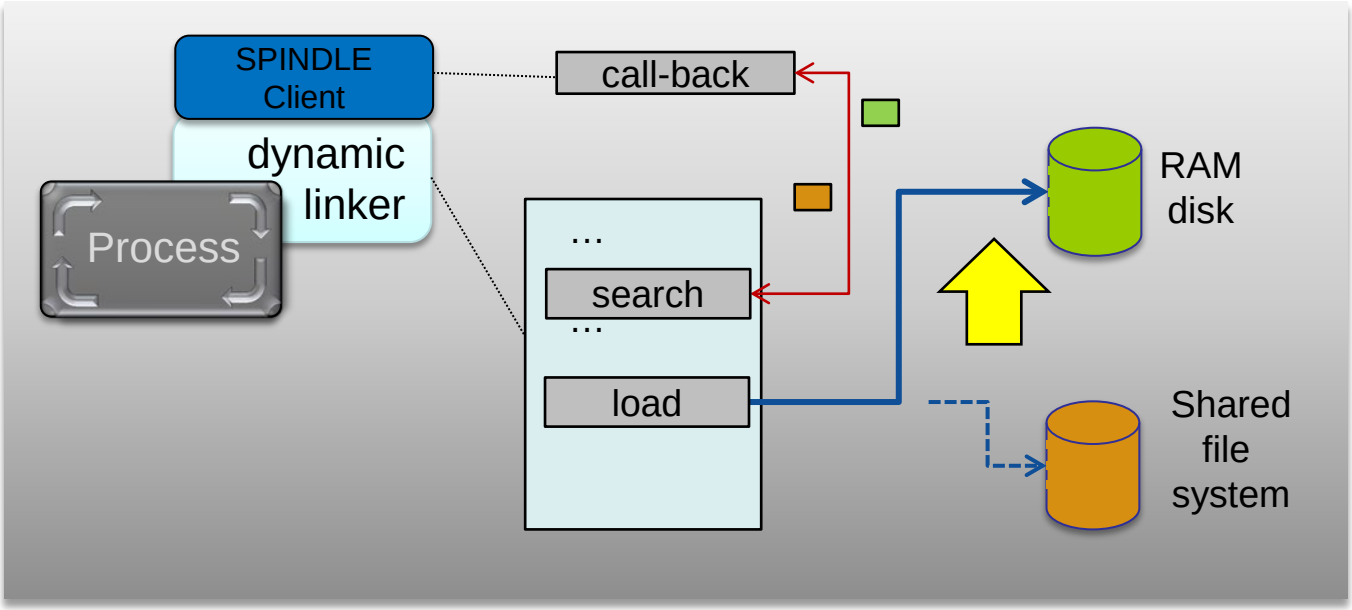
SPINDLE Components



→ Transparent user-space solution
- → SPINDLE Client
- → SPINDLE Server
- → Overlay Network

SPINDLE Client intercepts Dynamic Loader transparently

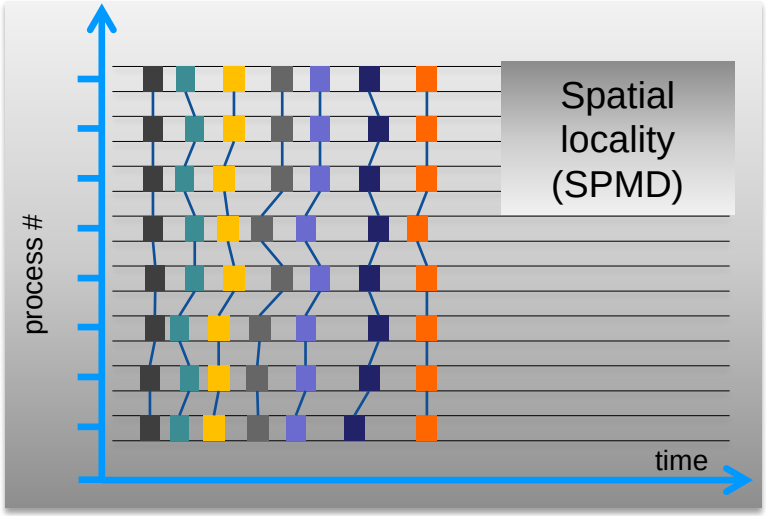
- Interception: rtld-audit interface of GNU linker
  - User-space definition of call-back functions
  - Redirect remote file system loads to a local RAM disk



SPINDLE Server Communication takes Advantage of Locality

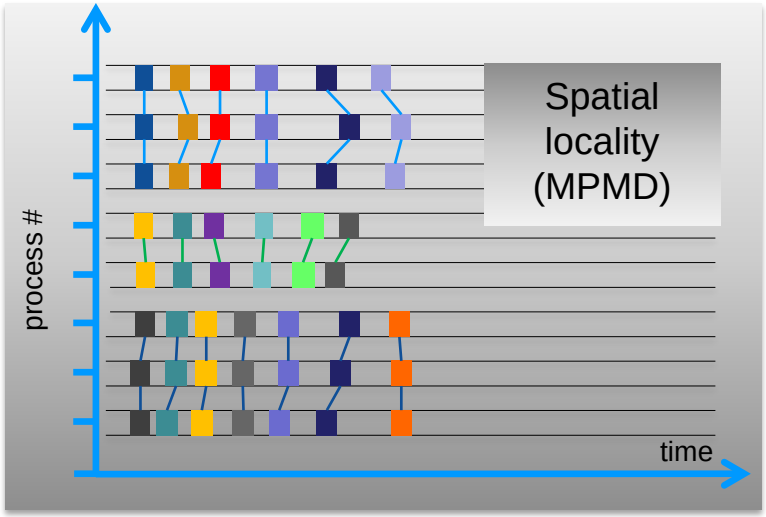
Spatial locality given by **SPMD** model

- Similar load sequence on all processes
- Push model broadcasts data immediately after first request



MPMD model

- Different load sequences on processes
- Pull model send data on request
- Often sets of SPMD groups

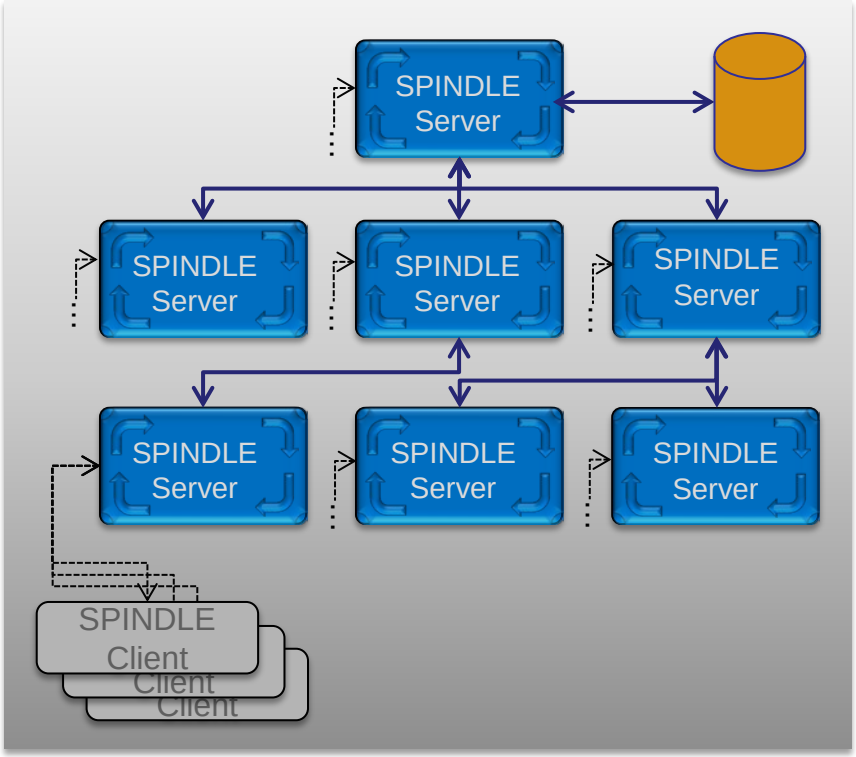


→ **SPINDLE supports Push and Pull models**

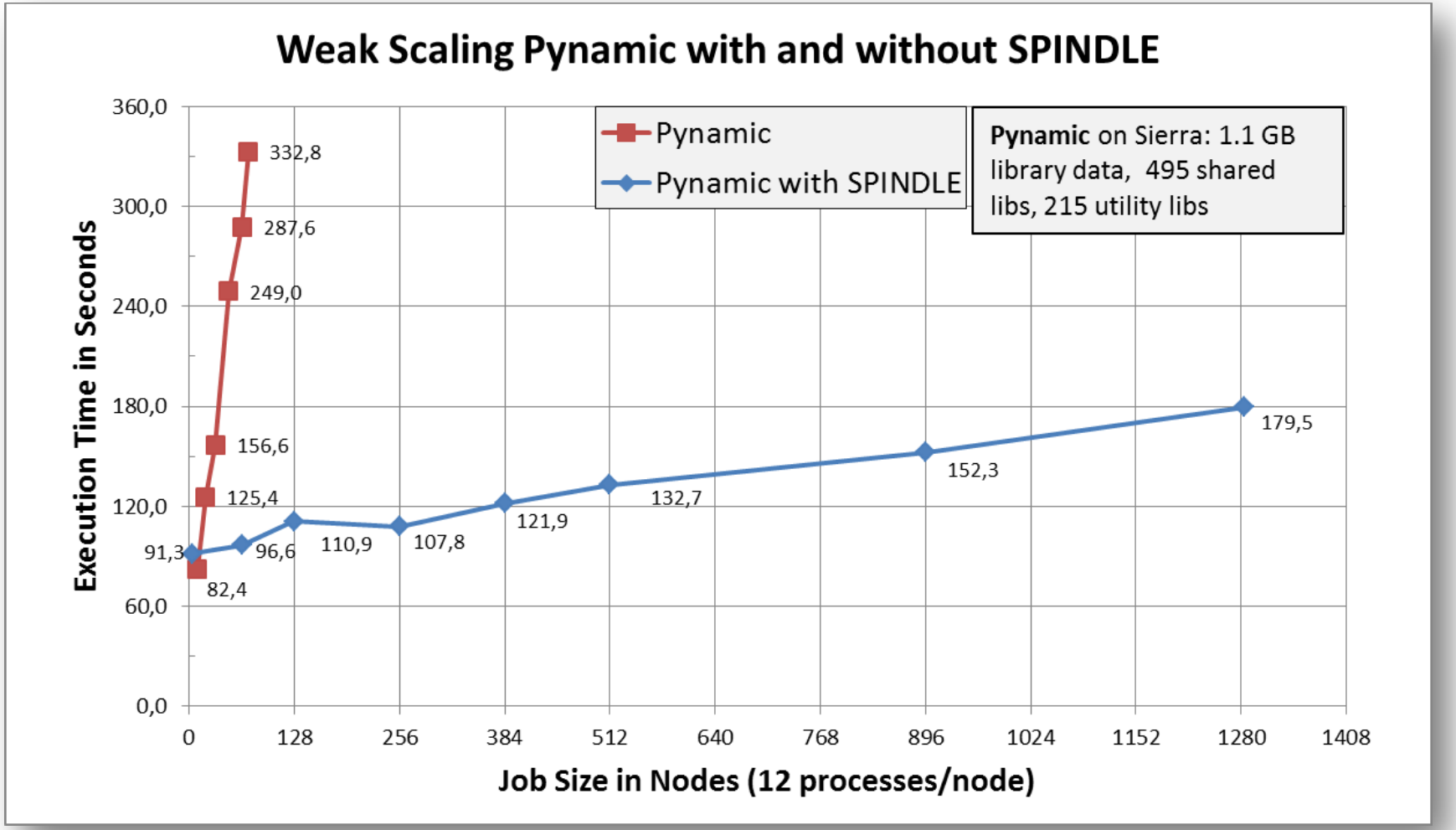
SPINDLE's Network Topology

- Push-Model → Tree topology
  - Root node is responsible for file system I/O
  - Limits number of message hops: log(n) *(n = # spindle server)*

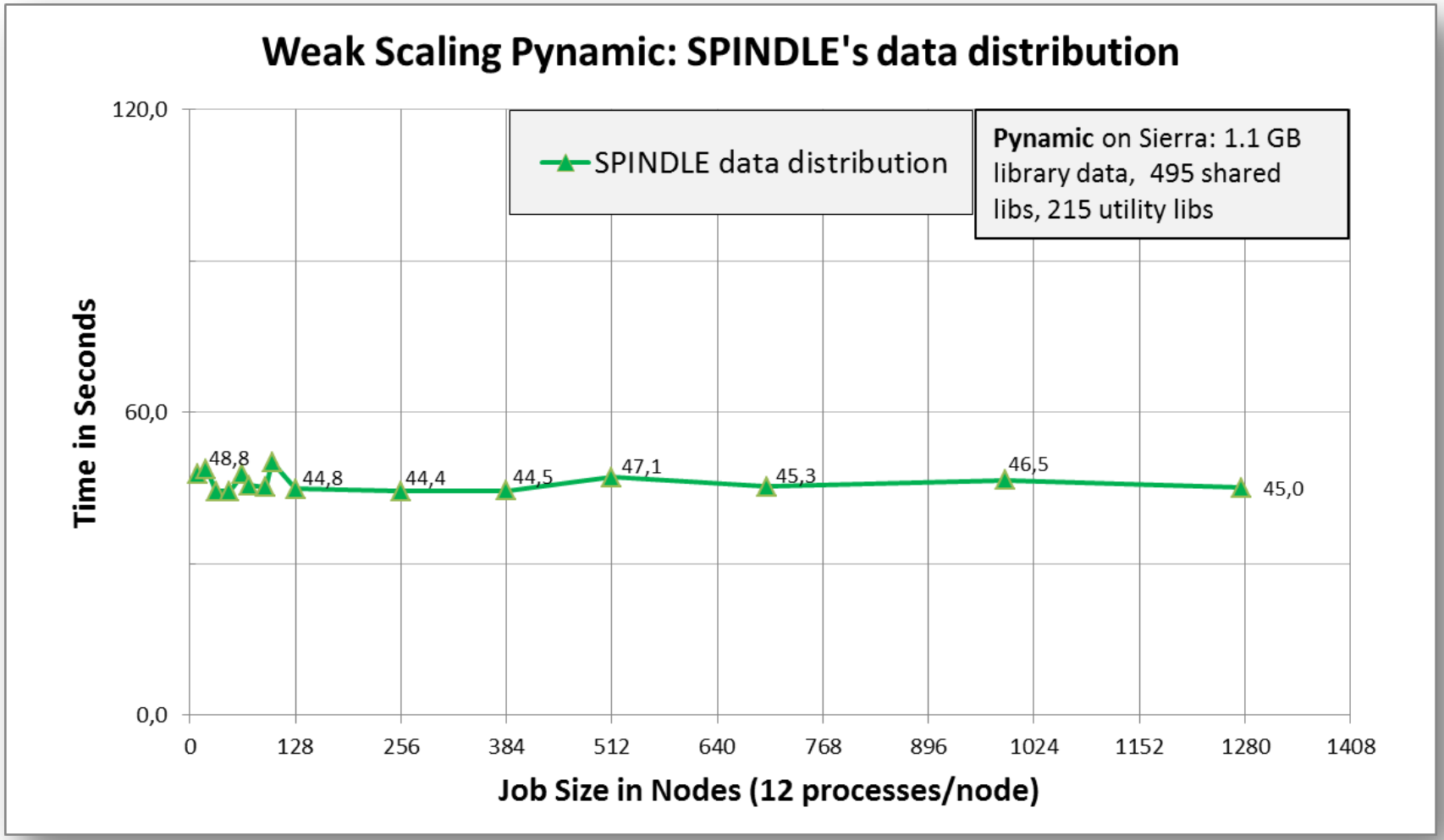
- Overlay network
  - Needed due to absence of system communication layer (MPI, …) at startup time
  - Using COBO, a scalable communication infrastructure built for MPI



SPINDLE's Performance

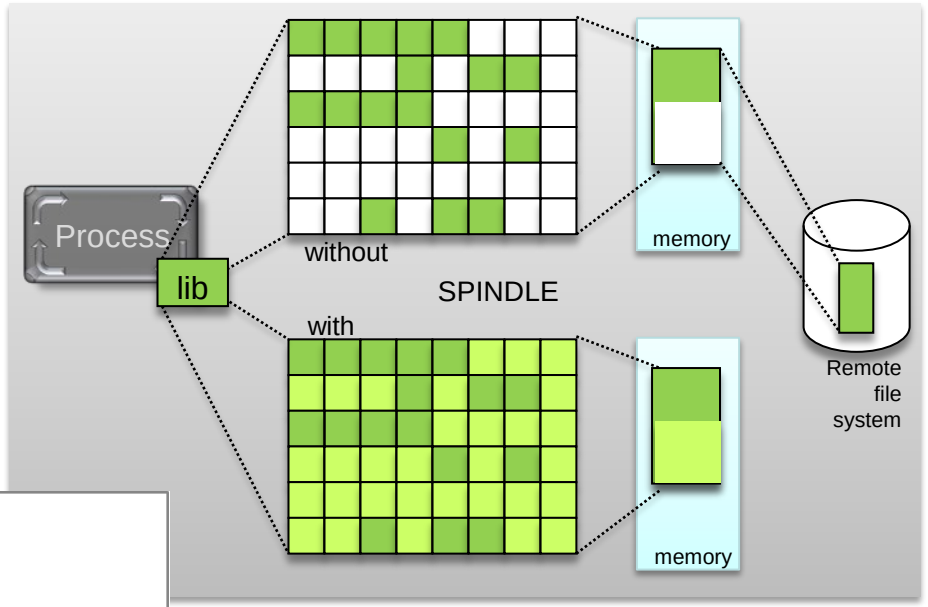


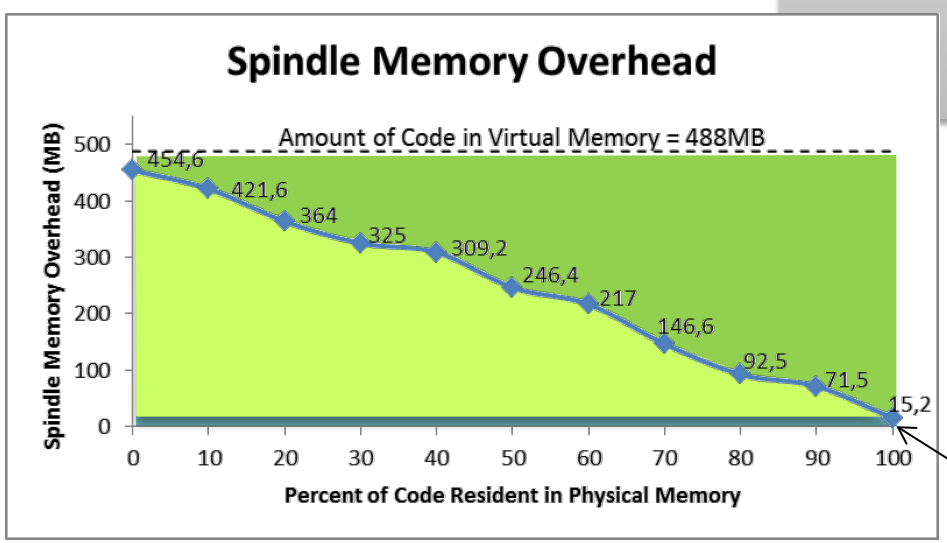
Constant Overhead of SPINDLE's Data Distribution



SPINDLE's Memory Footprint

- SPINDLE loads libraries to node-local RAM disk
- All pages are in memory
- No page reload during run-time (less OS-noise)





used pages

unused pages (in RAM disk)

unused pages (on remote FS)

SPINDLE overhead: < 15 MB

Launching SPINDLE

- SPINDLE wrapper call:

```
% spindle srun -n 512 myapp.exe <args>
```

- Executable is not modified
- SPINDLE scalably loads:
  - Library files (from dependencies and dlopen)
  - Executable
  - Python .py/.pyc/.pyo files
  - exec/execv/execve/… call targets
- Can follow forked processes
- Integrated with LaunchMON

Conclusion

- Loading of dynamic applications at large scale
  - Limited scalability on large HPC systems
  - File access storm → denial-of-service attack to file system
- SPINDLE
  - Extends dynamic loading by intercepting the dynamic loader through the auditing interface
  - Implements overlay network of file-cache servers for sharing location info, libraries, and Python files
  - Provides scalable and transparent environment for loading dynamic application
- Evaluation of SPINDLE shows loading with no disruption for
  - Pynamic benchmark on Sierra up to 15,312 MPI tasks with constant overhead for SPINDLE data distribution

Availability & Outlook

- Availability of SPINDLE
  - GitHub: https://github.com/hpc/Spindle
  - Documentation: https://scalability.llnl.gov/spindle
  - Licence: GNU Lesser General Public License (LGPL)
  - Build: Configure and Libtool, Test Environment
  - Version: 0.7.1
- SPINDLE's next steps
  - Porting, optimization and customization for broader range of HPC systems (e.g. IBM Blue Gene)
  - Tighter integration into various HPC system software (resource manager software systems)

SPINDLE's is a stepping stone on the way to a massively parallel OS/runtime loading service for future exascale systems